[illegible]

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/477,688	01/06/2000	STEPHEN ANTHONY EDWARDS	1000/5	9589

35795 7590 02/25/2004

JONATHAN T. KAPLAN
ATTORNEY AT LAW
140 NASSAU STREET
NEW YORK, NY 10038-1501

EXAMINER	
ALI, SYED J	
ART UNIT	PAPER NUMBER
2127	

DATE MAILED: 02/25/2004 19

Please find below and/or attached an Office communication concerning this application or proceeding.

PTO-90C (Rev. 10/03)

[illegible]

Office Action Summary	Application No. 09/477,688	Applicant(s) EDWARDS, STEPHEN ANTHONY	
	Examiner Syed J Ali	Art Unit 2127	

-- The MAILING DATE of this communication appears on the cov r sh t with the correspondenc address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on 12 January 2004.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) 1-12 is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) 1-12 is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on 07 July 2003 is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)

2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)

3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date _____.

4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .

5) ☐ Notice of Informal Patent Application (PTO-152)

6) ☐ Other: _____.

DETAILED ACTION

1. A request for continued examination under 37 CFR 1.114, including the fee set forth in 37 CFR 1.17(e), was filed in this application after final rejection. Since this application is eligible for continued examination under 37 CFR 1.114, and the fee set forth in 37 CFR 1.17(e) has been timely paid, the finality of the previous Office action has been withdrawn pursuant to 37 CFR 1.114. Applicant's submission filed on January 12, 2004 has been entered.

2. This office action is in response to Amendment B, paper number 18, which was filed January 12, 2004. Claims 1-12 are presented for examination.

3. The text of those sections of Title 35, U.S. code not included in this office action can be found in a prior office action.

Drawings

4. In view of Applicant's consultation with the Office of Patent Legal Administration, the objection to the drawings is withdrawn.

Claim Rejections - 35 USC § 103

5. Claims 1-12 are rejected under 35 U.S.C. 103(a) as being unpatentable over Lin (previously cited) in view of Nilsen et al. (previously cited) (hereinafter Nilsen).

As per claim 1, Lin discloses a method performed by a data processing system having a memory, comprising the steps of:

inputting a CCFG (pg. 213, Figs. 2[a-b], 3[a-b], wherein the a CCFG is interpreted to be a collection of nodes and edges indicating program flow control distributed among a plurality of threads, and the figures in Lin depict two processes running in parallel, as claimed);

scheduling the CCFG to produce a scheduled CCFG (pg. 215, "Our software synthesis method aims to produce, as intermediate output, plain C code that retains a high degree of parallelism so that the subsequent processor-specific code generation step can produce efficient executable machine code for the target processor", wherein the generation of compilable code is done specifically for the purpose of executing it on a processor, and thus inherently must be scheduled for execution);

selecting a first node of the scheduled CCFG (pg. 213, Figs 2[a-c], elements p1, p2, wherein copies of the nodes p1 and p2 are taken from the respective CCFGs in Figs 2[a-b] and used as similar nodes in the SCFG of Fig 2[c]);

producing a first copy of the first node for an SCFG (see citation and remark above); and

coupling, if a first thread of the first node is suspended, between a second node of the SCFG of a second previously running thread and the first copy, a first context switch (Fig. 2[c], element c2, wherein the decision point c2 is coupled between the two concurrently executing threads, and is a point at which the flow of execution may switch contexts).

Nilsen discloses the following limitations not shown by Lin, specifically wherein the context switch saves a second state, of the second previously running thread, into a state variable dedicated to the second previously running thread (col. 37, lines 60-67, "When a task is preempted, the dispatcher saves this information into the thread's state variables. Upon resumption, the thread restores these state variables from the saved thread information").

It would have been obvious to one of ordinary skill in the art to combine Lin with Nilsen since the method of combining concurrent control flow graphs into sequential control flow graphs disclosed by Lin takes into account context switching between separate processes or threads, but fails to specify exactly how the code translation is generated, or what features are implemented therein. Rather, the preliminary ideas behind the construction of Petri net representations of sequential control flow graphs is disclosed, while the features related to scheduling and optimization are left up to the developer (pg. 215, "It is not the intention of this paper to discuss in details the different possible scheduling heuristics. The interested reader can refer to [4, 6] for a survey of example techniques"). The papers that Lin refers to presumably offer several scheduling techniques that can be used for the sequential control flow graph, yet the possible scheduling techniques are not limited to those references. Nilsen offers a technique for protecting the state of a thread during a context switch or a preemption by saving the information related to the suspended thread in a state variable and utilizing the information in the state variable upon resumption of the stopped thread, which would allow the thread and it's shared resources to remain in a consistent state during any context switches or synchronization.

As per claim 2, Nilsen discloses the method of claim 1, wherein the first context switch is comprised of code that saves a state of a thread being suspended in a state variable (col. 37, lines 60-67, "When a task is preempted, the dispatcher saves this information into the thread's state variables. Upon resumption, the thread restores these state variables from the saved thread information").

Lin discloses resuming another thread by performing a multiway branch on a state variable for a thread being resumed (pg. 213, Figs. 2[c] and 3[c], wherein referring to Fig. 2[c], the transition point c2 is an exemplary instance of where a context switch may take place).

It would have been obvious to one of ordinary skill in the art to combine Lin with Nilsen for reasons discussed above in reference to claim 1.

As per claim 3, Lin discloses the method of claim 1, wherein the translation of the CCFG into the SCFG produces, for each node of the CCFG, at most one corresponding node in the SCFG (pg. 213, "concurrent processes can be composed via parallel composition...[,which is] essentially a Cartesian product of the two Petri net processes along common labeled actions"). Specifically, the combination of more than one concurrent process is translated to a single sequential process, and the graphs are joined at places where data dependencies between the concurrent processes indicate that a context switch should take place. Therefore, each node in the original concurrent graph maps to at most one node in the sequential graph, the exception being where two states coincide and would thus collapse into one node. The more common occurrence is that two transitions would coincide and that would indicate where a context switch should take place, in which case the nodes map on a one-to-one basis.

As per claim 4, Lin does not specifically disclose the method of claim 1, wherein the step of scheduling further comprises a topological sort for determining the scheduled augmented CCFG.

"Official Notice" is taken that the use of topological sorts is well known and expected in the art, and would have been an obvious modification to Lin. Specifically, Lin discloses an ordered graph, with nodes used to refer to specific variable states and edges to traverse between those states (pg. 212-213, Figs. 2[a-c], 3[a-c], wherein based on the code on pg. 212, a graphical representation is created in figures 2[a-b]).

A topological sort is well known as a way of ordering nodes, such that if a transition occurs between two nodes, then based on how that transition is represented, it is known what order the nodes occur in the graph. A common way of expressing this is that if an edge exists such that edge(u, v) is in the graph and u and v are nodes in the graph, then u comes before v in the ordering of the graph. This can be found in any number of programming guides, and an example is presented in the Boost Graph Library on the Boost C++ Library website (www.boost.org/libs/graph/doc/topological_sort.html). It would have been obvious to one of ordinary skill in the art to use a topological sort to determine the ordering of the scheduled augmented CCFG since the technique is well known, other programming methods that have been previously devised can be used in accordance with the sorting technique. Specifically, defining the graph with similar data structures would allow a programmer a multitude of predefined methods to operate on the data therein, depending on the specific needs of each individual system. Furthermore, Lin makes mention of how various features of the sequential control flow graph must be mapped out in order to generate the necessary code. This is done via a pre-ordering method, which represents the flow of control in various data structures, including a representation of all the "reachable nodes". A topological sort would have been an obvious method of generating such a set of "reachable nodes" by defining all the nodes of the graph as well as the state transitions within a single data structure.

As per claim 5, Lin discloses the method of claim 1, wherein an execution of the SCFG comprises translation of the SCFG into a programming language (pg. 213-216, §4.2, "Our software synthesis method aims to produce, as intermediate output, plain C code that retains a high degree of parallelism so that the subsequent processor-specific code generation step can produce efficient executable machine code for the target processor", wherein it is clear that execution of the sequential data flow model results from the compilation of the expansion into C code and execution of that executable for a specific processor).

As per claim 6, Lin discloses the method of claim 5, wherein the programming language is C (pg. 213-216, §4.2, "Our software synthesis method aims to produce, as intermediate output, plain C code that retains a high degree of parallelism so that the subsequent processor-specific code generation step can produce efficient executable machine code for the target processor").

As per claim 7, Lin discloses the method of claim 1, further comprising a step of translation of the SCFG into a programming language (pg. 213-216, §4.2, "Our software synthesis method aims to produce, as intermediate output, plain C code that retains a high degree of parallelism so that the subsequent processor-specific code generation step can produce efficient executable machine code for the target processor").

As per claim 8, Lin discloses the method of claim 7, further comprising a step of executing the programming language translation of the SCFG (pg. 213-216, §4.2, "Our software synthesis method aims to produce, as intermediate output, plain C code that retains a high degree of parallelism so that the subsequent processor-specific code generation step can produce efficient executable machine code for the target processor").

As per claim 9, Lin discloses the method of claim 1, wherein an execution of the SCFG comprises interpretation of the SCFG (pg. 213-216, §4.2, "Our software synthesis method aims to produce, as intermediate output, plain C code that retains a high degree of parallelism so that the subsequent processor-specific code generation step can produce efficient executable machine code for the target processor", wherein the translation into C code and generation of an executable therein amounts to interpretation of the SCFG resulting in an execution of the SCFG).

As per claim 10, Lin discloses a data processing system having a memory and capable of implementing the method of claim 1 (Abstract, wherein Lin is disclosed in conjunction with a computer system)

It would have been obvious to one of ordinary skill in the art to combine Lin with Nilsen for reasons discussed above in reference to claim 1.

As per claim 11, Lin discloses a computer program product comprising a computer usable medium having computer readable code embodied thereon and capable of implementing the method of claim 1 (Abstract, wherein Lin is disclosed in conjunction with a computer system and a method of optimizing software

It would have been obvious to one of ordinary skill in the art to combine Lin with Nilsen for reasons discussed above in reference to claim 1.

As per claim 12, Lin discloses a computer data signal embodied in a carrier wave and representing sequences of instructions which, when executed by a processor, cause performance of the method of claim 1 (Abstract, wherein Lin is disclosed in conjunction with a computer system, wherein the software synthesis method is implemented in software and carried out by electrical signals on a hardware level).

It would have been obvious to one of ordinary skill in the art to combine Lin with Nilsen for reasons discussed above in reference to claim 1.

Response to Arguments

6. Applicant's arguments filed January 12, 2004 have been fully considered but they are not persuasive.

7. On page 6, Applicant argues "*Lin <u>teaches away</u> from the use of thread-dedicated state variables by always encoding the state, of all concurrent threads, with the state a single program counter*".

While in the particular embodiment disclosed by Lin, a program counter is used to track the progress of each concurrent thread, this does not necessarily teach away from the use of a state variable or preclude other methods from being utilized. Rather, the use of a token in each concurrent thread functions in a similar fashion to the claimed state variable. It is Examiner's understanding that the token associated with each concurrent thread tracks the progress of the execution of that thread. Upon reaching a context switch, control is shifted to the other thread, with state information coming from the input received from the first thread as well as the state information stored within the token of the second thread. This method of context switching is very similar to the use of state variables, as exhibited by Nilsen. In particular, numerous situations may arise in a multithreaded environment that could require a context switch. These may include preemption, time slicing, blocking, synchronization, and locking techniques that may be associated with shared resources. It is of utmost importance that shared resources are protected against inconsistencies. Use of state variables is prevalent within multithreaded environments for preempted or otherwise suspended threads, one example of which is demonstrated by Nilsen.

8. On page 7, Applicant argues "*Note that even for the small number of states (or expansions) in the particular examples chosen by Lin, Lin notes the difficulties of generating code for each such state, due to concurrent processes sharing a single program counter, but offers no alternative*". Additionally, on page 8, Applicant argues "*Nilsen relates to the implementation of virtual machines and has no suggestion that its teachings could be utilized for the Lin method.*"

As noted above in number 7, the combination of Lin and Nilsen provides a way of switching contexts using thread-local state variables. Although Lin uses program counters, this does not preclude the use of other methods, and Lin further states that other methods of resource allocation can be used, and is at the discretion of the developer (pg. 215, "Although this pre-ordering step is closely related to the traditional scheduling problem [4, 6], we do not yet perform any detailed scheduling of instructions or any detailed resource allocation here"). Lin also states that the scheduling algorithm could be any of a number of methods (pg. 215, "It is not the intention of this paper to discuss in details the different possible scheduling heuristics. The interested reader can refer to [4, 6] for a survey of example techniques"). Lin does not seek to implement specific functionality related to the accessing of shared resources or the scheduling of threads. Rather, a method of generating a sequential flow of control that has portability across multiple processors is presented. Plain C code is generated that can then be optimized according to the specific processor. In the example of Nilsen, the processor is a virtual processor embedded with a virtual machine. Thus, the scheduling and resource allocation methods of Nilsen fit within the optimization portion of the Lin method.

9. On page 8, Applicant argues "*While topological sorts may be generally known, as are a great many other possible techniques that may be applicable to any given problem domain, applicant respectfully points out that the Examiner has presented no teaching that would suggest application of topological sort to the invention as claimed.*"

Examiner has added further discussion related to topological sorts, particularly the relevance to the pre-sorting technique used by Lin. Specifically, Lin presents a need to represent the "reachable nodes" of the control flow graph as well as representing directed paths from one node to another. A topological sort would provide one method of generating such a representation by ordering nodes such that state transitions and the order in which nodes appear could be represented within a single data structure that could be easily manipulated to suit the concurrency and synchronization needs associated with the sequential flow of control.

Conclusion

10. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Syed J Ali whose telephone number is (703) 305-8106. The examiner can normally be reached on Mon-Fri 8-5:30, 2nd Friday off.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Meng-Ai T An can be reached on (703) 305-9678. The fax phone number for the organization where this application or proceeding is assigned is 703-872-9306.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

[illegible]

Syed Ali
February 10, 2004

[illegible]

MENG-AL T. AN
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100